1. **BP神经网络**

## 1.1数据处理与分析

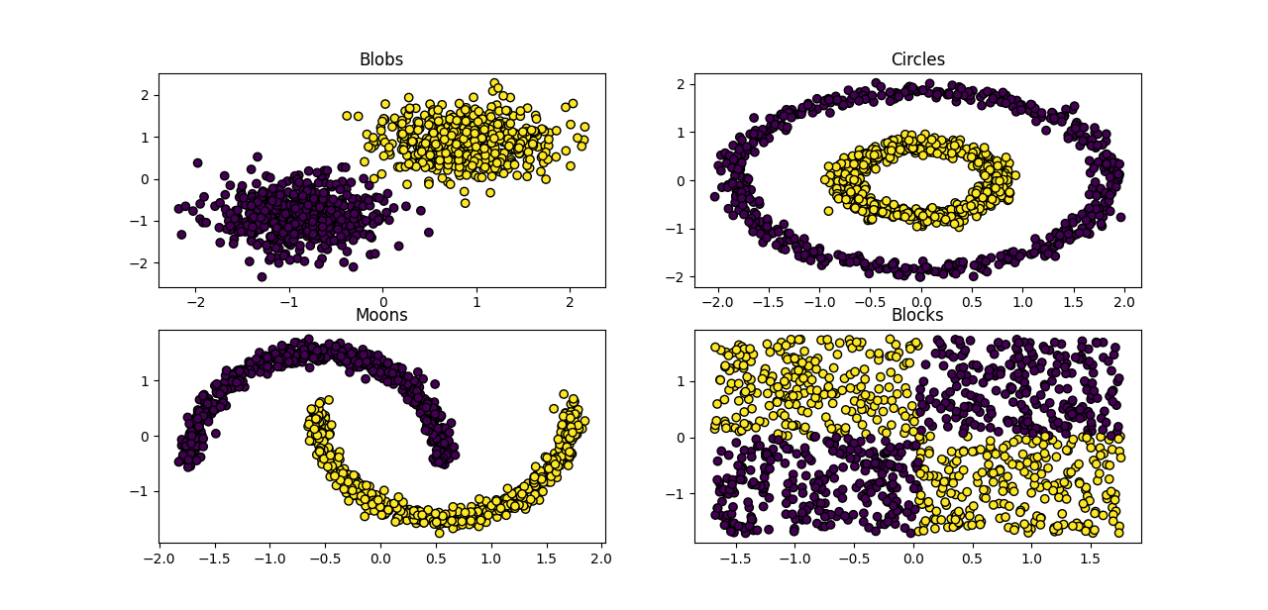
用给定的代码产生4个人工数据集：blobs，circles，moons和blocks，每个数据集都是二分类问题。将4个人工数据集的数据进行可视化展示了其数据点的分布和类别，如图1所示为4个数据集的散点图。

图 1 四个数据集散点图

Blobs数据集的两类数据呈现为两个中心不同的簇。这种分布形式较为简单明了，因为不同类别的数据在空间上有明显的分隔，使得模型能够较容易地区分它们。在训练过程中，模型可以较快地学习到两个簇之间的边界，并在测试时准确地进行分类。

Circles数据集的两类数据呈现为两个同心圆的形状。这种分布形式相对复杂一些，因为两个类别的数据在空间上有所重叠，但它们之间的边界是一个圆形。对于模型来说，需要学习到一种能够识别圆形边界的能力，以便在测试时准确地分类不同类别的数据。

Moons数据集的两类数据呈现为弯曲方向相反的月牙形状。这种分布形式更加复杂，因为不同类别的数据在形状和弯曲度上都有所不同，且它们之间的边界也是非线性的。这对模型提出了更高的要求，需要学习到一种能够识别非线性边界的能力，并适应不同类别数据在形状和弯曲度上的差异。

Blocks数据集的两类数据呈现为相互交错的方形区域。这种分布形式最为复杂，因为两个类别的数据在空间上高度重叠，且它们的边界不规则且难以识别。对于模型来说，需要学习到一种能够处理高度重叠和不规则边界的能力，以便在测试时准确地区分不同类别的数据。

## 1.2数学建模过程

**BP神经网络**

BP（反向传播）神经网络是一种广泛应用于各种分类和回归任务的多层前馈网络。其核心架构由输入层、隐藏层和输出层三部分组成，其中隐藏层可以根据问题的复杂性进行多层堆叠。每一层都包含一定数量的神经元，它们作为网络的基本处理单元，负责接收来自前一层神经元的信号，并进行一定的处理后再将信号传递给下一层。

在BP神经网络中，各层神经元之间的连接是定向的，即每个神经元只与相邻层的神经元进行连接，而同层内的神经元之间则没有连接。这种连接方式确保了信号在网络中的单向流动，即前向传播。此外，BP神经网络中的连接没有反馈回路，即每个神经元的输出不会直接影响其自身的输入，从而避免了网络中的递归和循环依赖。

BP神经网络的工作流程主要包括前向传播和反向传播两个阶段。在前向传播阶段，输入信号首先被输入层接收，并通过网络中的连接权值和神经元的激活函数逐层传播到隐藏层，并最终到达输出层，产生网络的输出信号。这个过程中，每个神经元都会根据其输入信号和自身的激活函数产生一个输出信号，该信号将被作为下一层神经元的输入信号继续传播。

然而，由于网络权值和阈值的初始值是随机设定的，因此网络在初始状态下的输出信号往往与期望信号存在较大的误差。为了减小这种误差，BP神经网络引入了反向传播阶段。在反向传播阶段，误差信号会沿着网络的连接通路反向传播，即从输出层逐层传回到输入层。在反向传播的过程中，网络会利用梯度搜索技术来不断调整各层神经元的权值和阈值，以使得网络的输出误差达到最小。这种调整过程基于梯度下降法的原理，即沿着误差函数对权值和阈值的负梯度方向进行搜索，以找到使误差函数达到最小值的权值和阈值。

通过前向传播和反向传播这两个阶段的不断迭代和优化，BP神经网络能够逐渐逼近复杂的非线性映射关系。在实际应用中，我们可以根据问题的具体情况选择合适的网络结构（如隐藏层的层数和神经元的数量）和激活函数，并通过训练数据来训练网络，使其能够准确地映射输入到输出之间的关系。训练完成后，我们就可以将训练好的BP神经网络应用于新的输入数据，并得到相应的输出结果。

**模型建立**

在利用BP神经网络进行学习和预测时，首先需要将数据集合理地划分为训练集和测试集。通常，为了获得更可靠的模型性能评估，将数据集的80%作为训练集，用于训练神经网络模型，而将剩余的20%作为测试集，用于评估模型的泛化能力。构建一个具有三层结构的神经网络，具体配置为[4,4,2]。这意味着输入层之后有两个隐藏层，每个隐藏层都包含4个神经元，而输出层有2个神经元，可能对应着两个不同的分类或预测结果。对于隐藏层的激活函数，选择sigmoid函数，因为它可以将任意输入值映射到0到1的范围内，并且对于中间层的输出来说，这种非线性映射能力是非常重要的。

在模型构建完成后，我们首先使用训练集数据对模型进行训练。训练过程中，网络会根据实际输出与期望输出之间的误差，通过反向传播算法调整各层神经元的权值和阈值，以最小化误差。这个过程会重复多次，直到模型的性能达到预设的标准或达到最大迭代次数。

训练完成后，我们使用之前预留的测试集数据对模型进行测试。测试过程中，我们将测试集数据输入到已经训练好的神经网络中，得到模型的预测输出。然后，我们将预测输出与测试集的真实数据进行比较，计算模型的准确率，并通过分类图等方式直观地展示模型的预测效果。

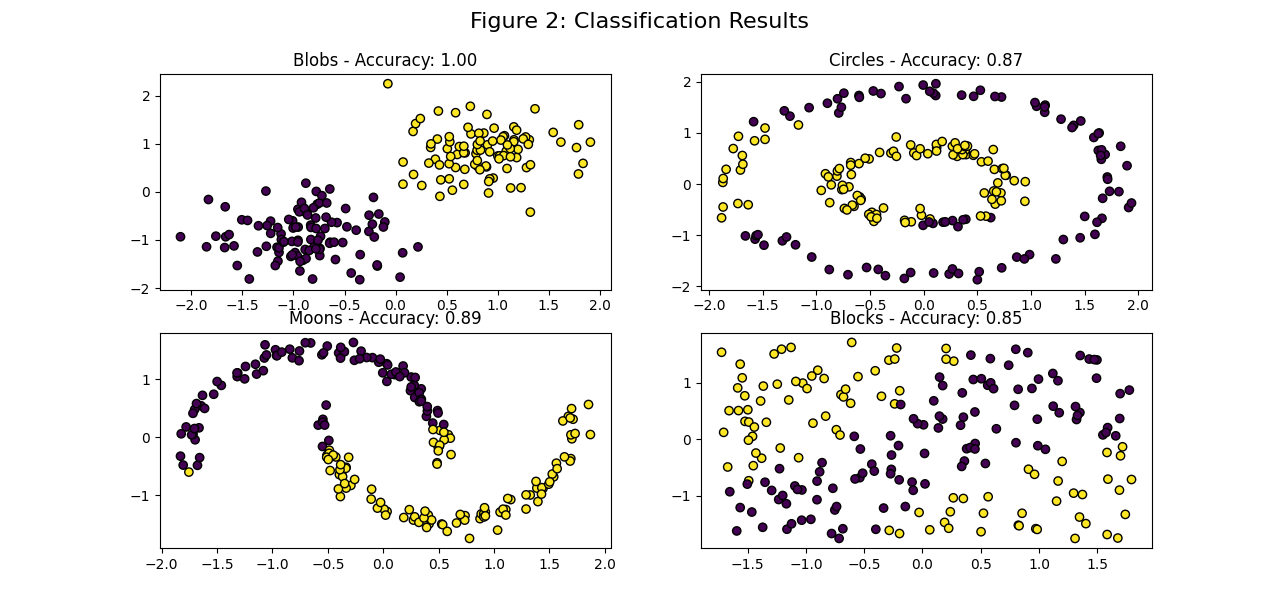
四个模型的准确率以及分类图如图2所示。

图 2 分类准确率及分类结果可视化图

从给出的图片中，我们可以深入观察并分析四个不同数据集（Blobs、Circles、Moons和Blocks）训练出的模型在测试集上的分类效果。

Blobs数据集。训练出来的模型在测试集上取得了100%的分类准确率，这显示出了模型极高的识别能力。这说明Blobs数据集中的样本特征分布清晰，模型能够准确地区分不同类别的样本，从而实现了很好的分类效果。

Circles数据集。训练出的模型在测试集上达到了87%的分类准确率。从可视化图中可以观察到，两个圆圈上的大部分点都被正确分类，但仍有少数点被错误分类。这可能是由于这些点位于两个圆圈的边界附近，特征较为模糊，导致模型难以准确判断其所属类别。

Moons数据集，训练出的模型在测试集上的分类准确率为89%。与Circles数据集类似，这一准确率也相对较高。然而，从可视化图中我们可以发现，在比较靠近的两个月牙的末端出现了少量错分的点。这可能是因为这些区域的样本特征相似度较高，模型在判断时产生了混淆。

Blocks数据集。训练出的模型在测试集上的分类准确率为85%，尽管准确率依然较高，但与其他数据集相比略低。从可视化图中可以看出，模型在判别中间带状区域时将其视为一类，而实际上该区域应该包含方形区域。这导致在边界处出现了较多的错分点。这可能是由于模型在训练过程中没有充分学习到方形区域与带状区域之间的特征差异，从而导致了误判。

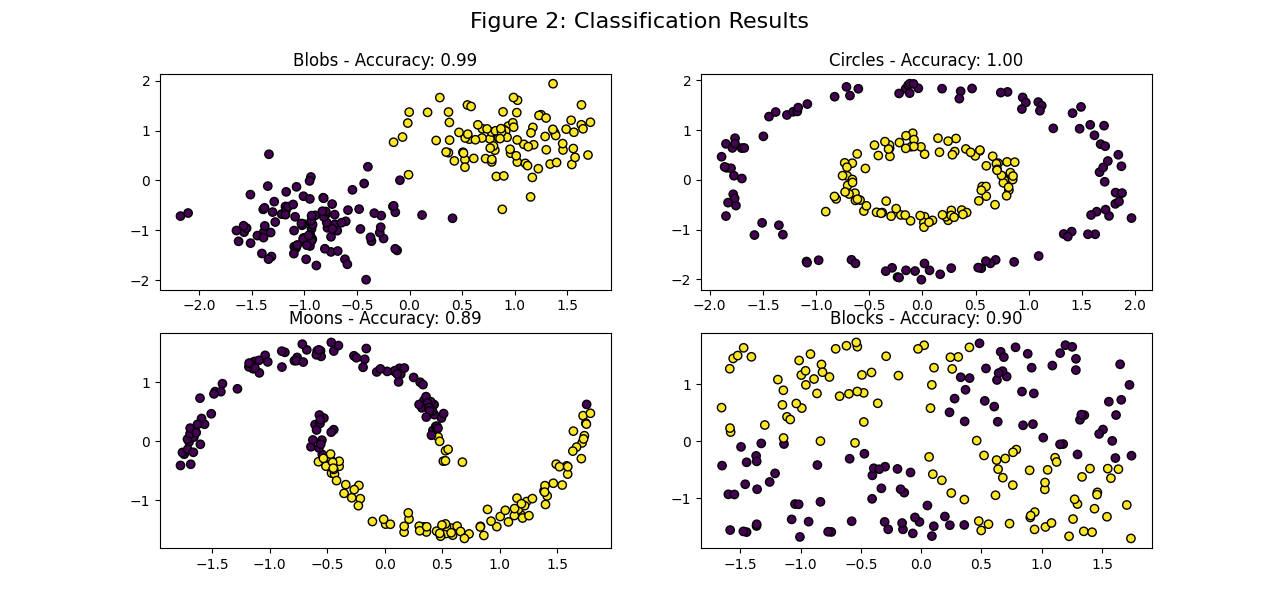
考虑增加神经元数量，考察模型的准确率。如图3所示，为神经元为10时的结果。

图 3 神经元数量为10时结果图

由图可知，在Circles数据集上增加神经元的数量时，模型的准确率有所提升，这可能是由于增加的神经元为模型提供了更强的非线性映射能力和更复杂的特征表示，从而使其能够更好地拟合数据集中的复杂模式。但对于Moons和Blocks数据集，情况却有所不同。尽管增加了神经元的数量，但这两个数据集上训练出的模型准确率却略有下降。这种现象表明，神经元数量的增加并不总是能提高模型的性能。实际上，当神经元数量过多时，模型可能会陷入过拟合的困境，即模型在训练集上表现良好，但在测试集上的性能却大打折扣。

对于Moons数据集，由于其数据分布具有非线性且复杂的边界，过多的神经元可能会导致模型过于复杂，从而难以捕捉到数据中的真正规律。同样地，Blocks数据集由于其图像中的区块形状、大小和颜色各异，也需要模型具备适当的复杂性来准确识别。如果神经元数量过多，模型可能会过度关注数据中的噪声和细节，而忽略了真正有用的特征。

因此，我们可以得出结论：神经元数量的增加并不一定能提高模型的准确率。在实际应用中，我们需要根据数据集的特点和任务的需求来选择合适的神经元数量。此外，我们还需要通过交叉验证、正则化等技术来防止模型过拟合，以确保模型在测试集上具有良好的性能。

## 1.3结果的评价与分析

**损失曲线**

损失曲线是机器学习和深度学习中一个至关重要的评估工具，它直观地展示了模型在训练过程中的损失函数值随训练轮次（或称为“epoch”）的变化情况。损失函数用于量化模型预测值与实际值之间的差距，因此，通过观察损失曲线的变化趋势，我们可以深入了解模型是否在学习，以及学习的速度和效率如何。

在理想情况下，随着训练的进行，模型通过不断调整其内部参数来优化其预测能力，这会导致损失值逐渐降低。如果损失曲线在训练过程中呈现出稳定下降的趋势并最终趋于平稳，那么这通常表明模型在训练集上的性能正在稳步提升，并且可能正在接近其最优解。这种情况下，我们可以对模型的未来表现持乐观态度，因为它正在以有效的方式学习数据中的模式。然而，损失曲线的变化并非总是如此理想。有时，损失值在训练过程中可能会出现较大的波动，或者始终无法降低到一个较低的水平。这些情况通常表明模型在训练过程中遇到了问题。例如，如果损失值在训练早期就迅速下降，但随后又急剧上升，这可能意味着模型出现了过拟合现象，即模型在训练集上表现良好，但在测试集或实际应用中表现较差。另外，如果损失值始终无法降低到一个较低的水平，这可能意味着模型存在欠拟合问题，即模型无法充分学习数据中的特征，导致预测效果不佳。

除了揭示模型的学习状态和问题外，损失曲线的变化趋势还可以用来预测模型的未来表现。如果损失值在训练后期仍然持续下降，那么这可能意味着模型还有进一步优化的空间，我们可以通过增加训练轮次、调整模型结构或优化算法等方式来进一步提升模型的性能。相反，如果损失值在训练早期就趋于平稳，那么这可能表示模型已经达到了其性能的上限，即使继续训练也很难再获得明显的提升。

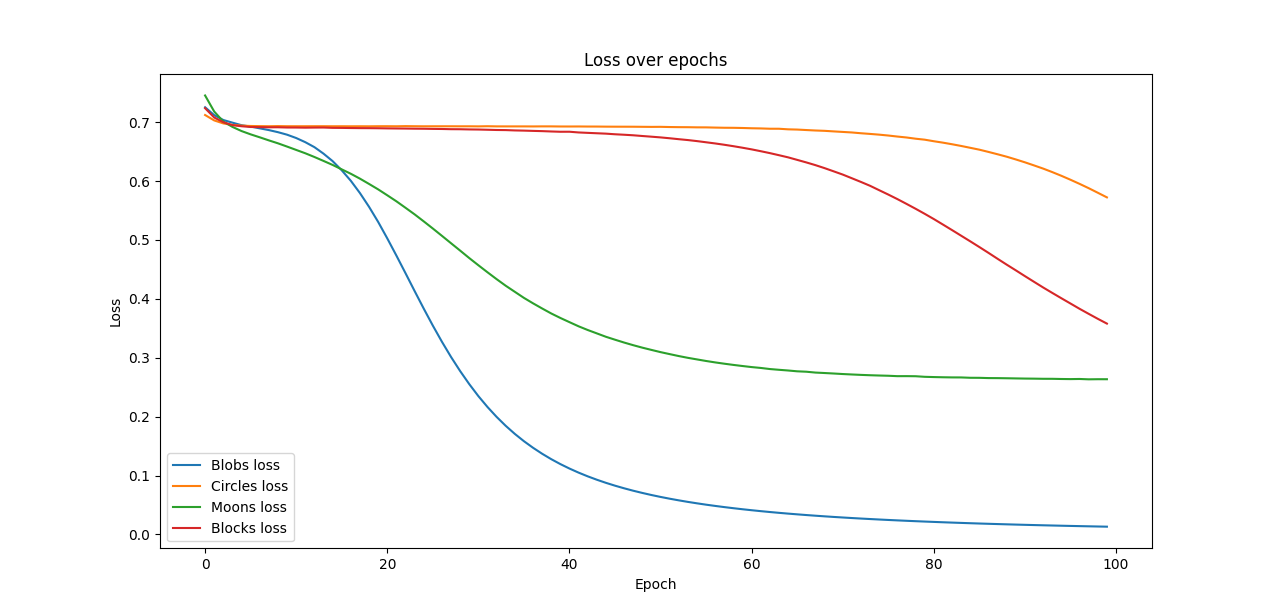
通过绘制多个训练历史的损失曲线，比较不同模型或不同数据集上的训练效果。如图4所示，为神经元为4时四个数据集损失曲线图。

图 4 损失曲线

对于Blobs和Moons这两个数据集，它们的损失曲线在训练初期就迅速下降，并在较短时间内进入了相对平缓的阶段。这一趋势清晰地表明，这两个数据集对应的模型在学习过程中很快就达到了一个较为稳定的性能水平，即它们较快地逼近了可能的性能上限。这通常意味着，针对Blobs和Moons数据集，模型的参数调整和优化过程比较高效，能够在较短时间内找到一个合适的解空间，使得模型能够较好地拟合数据集中的模式。

对于Circles和Blocks这两个数据集，它们的损失曲线在训练初期并没有像Blobs和Moons那样迅速下降，而是在经过了一段相对较长的时间后才开始逐渐降低。更为引人注目的是，即使在训练的中后期，这两个数据集的损失曲线依然呈现出明显的下降趋势，暗示着模型的性能还有进一步提升的空间。

这种差异表明，Circles和Blocks数据集的模型在训练过程中可能遇到了更大的挑战。或许是因为这两个数据集的数据分布更为复杂，或者是模型在初始阶段未能有效地捕捉到数据中的关键特征。这种持续的损失下降趋势都为我们提供了一个重要的线索：可能通过增加训练次数或采用更为复杂的模型结构，我们有可能进一步提升Circles和Blocks数据集上模型的性能。

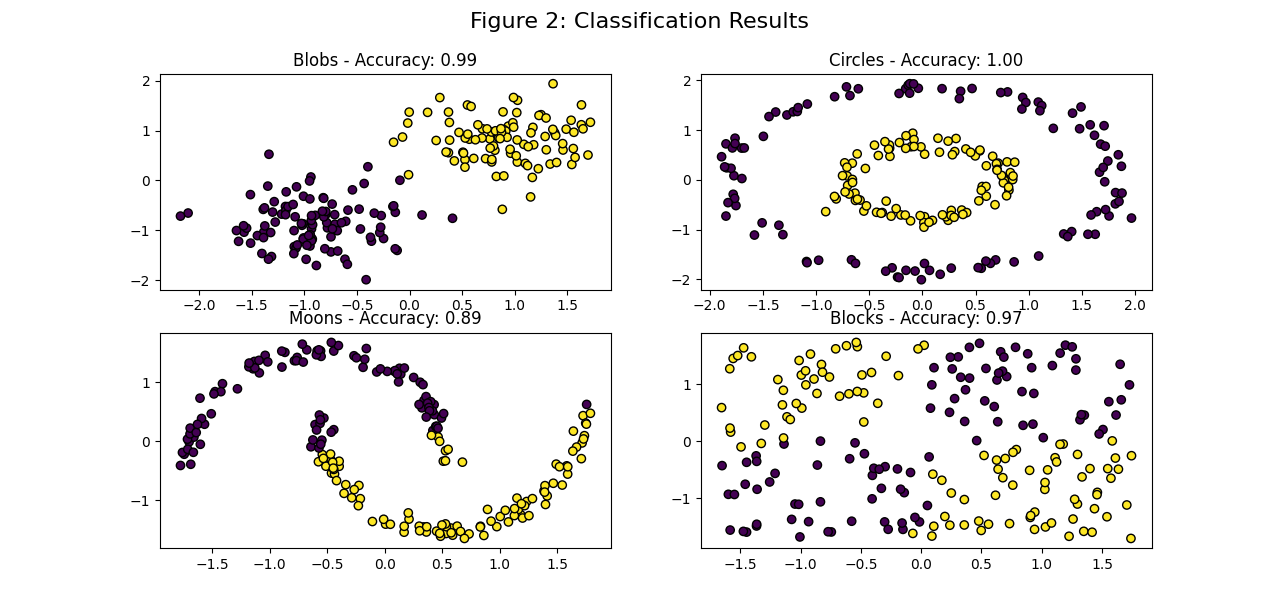
考虑增加训练次数，以得到训练更好效果的模型。如图5所示，为训练次数为300时的预测结果图。

图 5 训练次数为300时的预测结果

与图2中展示的训练初期的情况相比，我们可以进一步观察到Blobs和Moons数据集的准确率在后续的训练过程中基本没有发生显著的变化。这一现象强烈地表明，针对这两个数据集的模型已经在较短时间内达到了最优性能，其分类能力已经趋于稳定。因此，即便继续增加训练次数，也不太可能使这两个模型的性能得到进一步的提升。Circles和Blocks数据集的准确率在后续的训练过程中表现出了明显的提升。这说明，随着训练次数的增加，这两个模型能够更好地学习并识别数据集中的特征，从而提高分类的准确性。

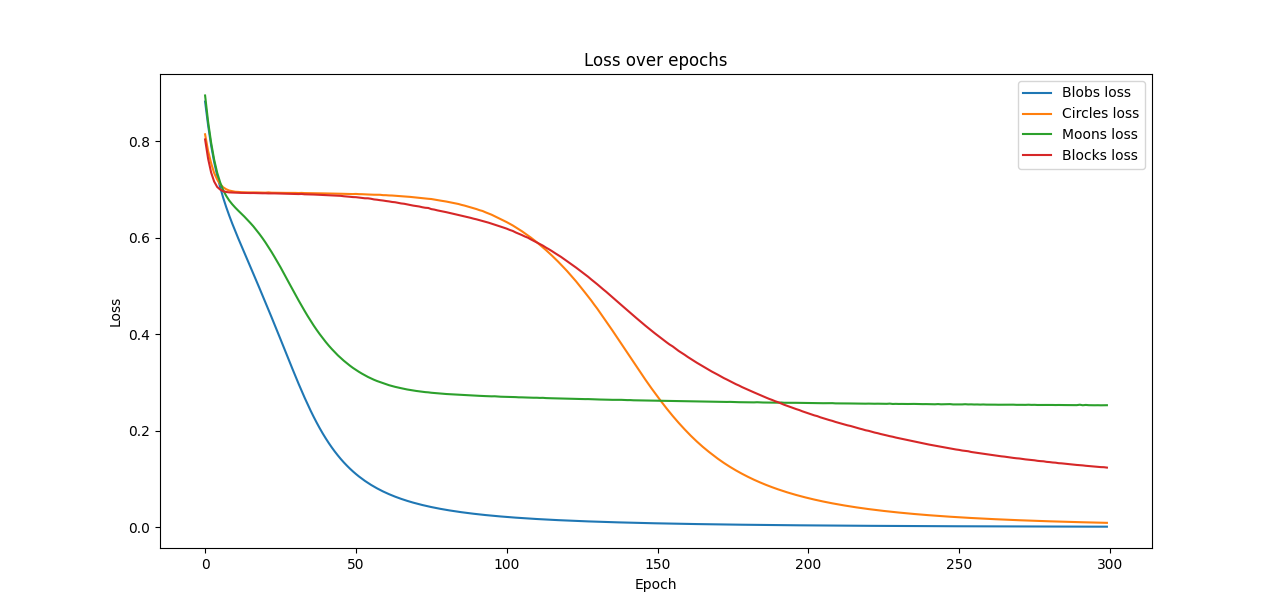
如图6所示，为训练次数为300时的损失曲线图。

图 6 训练次数为300时的损失曲线图

与图4进行对比，我们可以进一步验证前面关于Circles和Blocks数据集模型训练过程得出的结论。在图4中，我们观察到了Circles和Blocks数据集在训练过程中的损失曲线变化。经过250次的训练迭代后，这两个数据集的模型损失曲线开始趋于平稳，不再像早期阶段那样有显著下降。这一变化表明，模型在训练过程中已经逐渐找到了一个相对稳定的参数配置，使得其能够较好地拟合数据集的特征，并达到一个相对较低的损失值。

当训练轮次达到300次时，我们可以合理推断，这两个数据集的模型已经基本达到了其最优性能。因为损失曲线在250次迭代后已经趋于平稳，说明模型在后续的迭代中并没有再获得明显的性能提升。这表明，在现有的训练策略、模型结构和参数配置下，模型已经能够充分利用数据集中的信息，并达到了一个相对稳定的分类效果。

因此，我们可以得出结论，对于Circles和Blocks数据集，训练300次之后的模型已经基本达到了其最优性能。这意味着，如果我们继续增加训练轮次，不太可能再获得显著的性能提升，反而可能增加过拟合的风险和计算资源的消耗。因此，在实际应用中，我们可以考虑在训练轮次达到250次至300次之间时停止训练，以获得一个性能较好且稳定的模型。

## 1.4代码

|  |
| --- |
| import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import accuracy\_score  import matplotlib.pyplot as plt  import tensorflow as tf  from sklearn.datasets import make\_blobs, make\_circles, make\_moons  from sklearn.preprocessing import StandardScaler  def generateData(n):  np.random.seed(12046)  blobs = make\_blobs(n\_samples=n, centers=[[-2, -2], [2, 2]])  circles = make\_circles(n\_samples=n, factor=.4, noise=.05)  moons = make\_moons(n\_samples=n, noise=.05)  blocks = np.random.rand(n, 2) - 0.5  y = (blocks[:, 0] \* blocks[:, 1] < 0) + 0  blocks = (blocks, y)  # 由于神经网络对数据的线性变换不稳定，因此将数据做归一化处理  scaler = StandardScaler()  blobs = (scaler.fit\_transform(blobs[0]), blobs[1])  circles = (scaler.fit\_transform(circles[0]), circles[1])  moons = (scaler.fit\_transform(moons[0]), moons[1])  blocks = (scaler.fit\_transform(blocks[0]), blocks[1])  return blobs, circles, moons, blocks  def plotData(datasets, titles):  fig, axs = plt.subplots(2, 2, figsize=(12, 12))  for ds, title, ax in zip(datasets, titles, axs.flatten()):  ax.scatter(ds[0][:, 0], ds[0][:, 1], c=ds[1], cmap='viridis', edgecolor='k')  ax.set\_title(title)  plt.show()  def trainModelAndGetHistory(X\_train, y\_train, epochs=100, batch\_size=32):  model = tf.keras.models.Sequential([  tf.keras.layers.Dense(10, activation='sigmoid', input\_shape=(2,)),  tf.keras.layers.Dense(10, activation='sigmoid'),  tf.keras.layers.Dense(2, activation='sigmoid')  ])  model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])  history = model.fit(X\_train, y\_train, epochs=epochs, batch\_size=batch\_size, verbose=0)  return model, history  def plotLossCurves(histories, titles):  plt.figure(figsize=(16, 10))  for history, title in zip(histories, titles):  plt.plot(history.history['loss'], label=title + ' loss')  plt.title('Loss over epochs')  plt.xlabel('Epoch')  plt.ylabel('Loss')  plt.legend()  plt.show()  def trainAndPlot(datasets, titles):  fig, axs = plt.subplots(2, 2, figsize=(12, 12))  fig.suptitle('Figure 2: Classification Results', fontsize=16)  histories = []  for ds, title, ax in zip(datasets, titles, axs.flatten()):  X, y = ds  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  model, history = trainModelAndGetHistory(X\_train, y\_train)  histories.append(history)  y\_pred = np.argmax(model.predict(X\_test), axis=1)  accuracy = accuracy\_score(y\_test, y\_pred)  scatter = ax.scatter(X\_test[:, 0], X\_test[:, 1], c=y\_pred, cmap='viridis', edgecolor='k', marker='o')  ax.set\_title(f"{title} - Accuracy: {accuracy:.2f}")  # Plot loss curves  plotLossCurves(histories, titles)  n\_samples = 1000  datasets = generateData(n\_samples)  titles = ['Blobs', 'Circles', 'Moons', 'Blocks']  plotData(datasets, titles)  trainAndPlot(datasets, titles) |